**Connect Four Game with Minimax Algorithm and Alpha-Beta Pruning**

**Introduction:**

This project implements a two-player Connect Four game, where one player is human and the other player is controlled by an AI using the Minimax algorithm with Alpha-Beta pruning. The game is played on a 7x6 grid, and the objective is to connect four discs in a row horizontally, vertically, or diagonally. The AI is designed to consider strategic positions and block potential human wins, using a static evaluation function to assess game states.

**PEAS - Data Structures and Fringes that Define the Agent Environment**

The environment is represented as a 7x6 grid using a 2D array. The grid is initialized with zeros, representing empty spaces. Each player (human or AI) is assigned a unique disc (Player 1 as 1 and Player 2 as 2), which they drop into one of the columns on the grid. The following functions were used to define actions and the environment:

* **create\_grid()**: Initializes an empty grid.
* **drop\_disc(grid, row, col, disc)**: Drops a player's disc into the chosen column at the lowest available row.
* **is\_valid\_location(grid, col)**: Checks whether a column has space for a new disc.
* **get\_next\_open\_row(grid, col)**: Finds the next available row in a chosen column.
* **print\_grid(grid)**: Prints the grid to the console in a visually intuitive format.
* **is\_board\_full(grid)**: Checks if the grid is full, which results in a draw.

The environment (grid) is dynamically updated with each move, and the current state is displayed after every turn.

**Minimax Algorithm Implementation**

The **Minimax algorithm** is used to determine the best possible move for the AI. It simulates all possible future game states up to a fixed depth (in this case, 3 levels) and assigns scores to each possible move. The goal of the AI is to maximize its score while minimizing the human player's score, considering potential future outcomes.

* **minimax(grid, depth, alpha, beta, maximizingPlayer)**: This function recursively evaluates game states up to the defined depth. The **maximizing player** (AI) tries to maximize its score, while the **minimizing player** (human) tries to minimize the AI's score.
* The algorithm prunes branches where a better outcome is not possible using **alpha-beta pruning** to improve efficiency.
* **Alpha** represents the best score the maximizing player (AI) can achieve.
* **Beta** represents the best score the minimizing player (human) can achieve.

The Minimax algorithm evaluates all valid locations for the AI, simulating future human and AI moves, and returns the best column for the AI's next move.

**Alpha-Beta Pruning Implementation**

Alpha-Beta pruning is integrated within the Minimax algorithm to cut off branches of the search tree that are not useful, reducing computation time. This is done by keeping track of two values:

* **Alpha**: The best value the maximizer can guarantee.
* **Beta**: The best value the minimizer can guarantee.

Whenever a branch's potential score is worse than an already evaluated branch, it is pruned to avoid unnecessary calculations.

**Static Evaluation Function**

The static evaluation function is used to score the current game state for the AI. It assigns points based on strategic patterns, such as:

* **Center column preference**: Favouring the center column as it provides more potential winning connections.
* **Three-in-a-row patterns**: The AI prefers patterns that are close to forming four-in-a-row. For example, a window of three discs and one empty space gets a higher score.
* **Blocking opponent's moves**: The AI also attempts to block the human player from connecting four discs in a row, especially when the human has three consecutive discs.

This evaluation helps the AI make strategic decisions, rather than just reacting to immediate threats.

* **evaluate\_window(window, disc)**: Scores a window of four consecutive slots based on how favourable it is for the AI or the human.
* **score\_position(grid, disc)**: Aggregates the scores from all possible windows on the grid, including rows, columns, and diagonals.

**Start the Game - Human vs. AI Interaction**

The main game loop allows the human and AI to take turns, alternating between each move. The human player inputs a column number (0-6) to drop their disc, while the AI uses the Minimax algorithm to determine the best column to place its disc.

Key game features:

* **Alternating Turns**: The game switches between the human and AI after each move.
* **Move Validation**: The game ensures that a player cannot select a full column, and it prompts the human player to reselect if necessary.
* **Win Checking**: After each move, the game checks if the player has connected four discs in a row (horizontally, vertically, or diagonally). If a player wins, the game announces the winner and terminates.
* **Draw Condition**: If the grid is full and no player has won, the game ends in a draw.

The game flow is handled using the play\_game() function, which manages turn-taking and game state updates.

**Sample Game Scenario:**

1. The grid is initialized as empty, and the human player is prompted to select a column.
2. The human drops a disc into column 3.
3. The AI evaluates the board and selects a column based on the Minimax algorithm.
4. The game alternates between the players, displaying the grid after each move.
5. If a player connects four discs, the game ends and the winner is announced.
6. If the grid is filled without any player winning, the game declares a draw.

**Conclusion:**

The project implements a fully interactive Connect Four game where a human competes against an AI controlled by the Minimax algorithm. Alpha-Beta pruning optimizes the decision-making process, and the static evaluation function ensures that the AI makes strategic decisions. The game alternates turn between the human and AI, displaying the board state after each move, and ends when either a player wins or the grid is full.